pydisque Documentation

Release 0.1.1

ybrs

Jul 11, 2017

Contents

Create a new Disque client by passing a list of nodes:

```
from pydisque.client import Client
c = Client(["127.0.0.1:7711", "127.0.0.1:7712", "127.0.0.1:7713"])
c.connect()
```

If it can't connect to first node, it will try to connect to second, etc.., if it can't connect to any node, it will raise a redis.exceptions.ConnectionError as you can imagine.

Now you can add jobs:

```
c.add_job("test_queue", json.dumps(["print", "hello", "world", time.time()]),␣
↪timeout=100)
```

It will push the job "print" to the queue "test_queue" with a timeout of 100 ms, and return the id of the job if it was received and replicated in time. If it can't reach the node - maybe it was shutdown etc. - it will retry to connect to another node in given node list, and then send the job. If there is no avail nodes in your node list, it will obviously raise a ConnectionError

Then, your workers will do something like this:

```
while True:
    jobs = c.get_job(['test_queue'])
    for queue_name, job_id, job in jobs:
        job = json.loads(job)
        print ">>> received job:", job
        c.ack_job(job_id)
```

Contents:

class pydisque.client.**Client**(*nodes=None*)

Client is the Disque Client.

You can pass in a list of nodes, it will try to connect to first if it can't then it will try to connect to second and so forth.

Example

```
>>> client = Client(['localhost:7711', 'localhost:7712'])
>>> client.connect()
```

ack_job(**job_ids*)

Acknowledge the execution of one or more jobs via job IDs.

ACKJOB jobid1 jobid2 ... jobidN

Parameters **job_ids** – list of job_ids

add_job(*queue_name*, *job*, *timeout=200*, *replicate=None*, *delay=None*, *retry=None*, *ttl=None*, *maxlen=None*, *async=None*)

Add a job to a queue.

ADDJOB queue_name job <ms-timeout> [REPLICATE <count>] [DELAY <sec>] [RETRY <sec>] [TTL <sec>] [MAXLEN <count>] [ASYNC]

Parameters

- **queue_name** – is the name of the queue, any string, basically.
- **job** – is a string representing the job.
- **timeout** – is the command timeout in milliseconds.

- **replicate** – count is the number of nodes the job should be replicated to.
- **delay** – sec is the number of seconds that should elapse before the job is queued by any server.
- **retry** – sec period after which, if no ACK is received, the job is put again into the queue for delivery. If RETRY is 0, the job has an at-most-once delivery semantics.
- **ttl** – sec is the max job life in seconds. After this time, the job is deleted even if it was not successfully delivered.
- **maxlen** – count specifies that if there are already count messages queued for the specified queue name, the message is refused and an error reported to the client.
- **async** – asks the server to let the command return ASAP and replicate the job to other nodes in the background. The job gets queued ASAP, while normally the job is put into the queue only when the client gets a positive reply.

Returns job_id

connect()
Connect to one of the Disque nodes.

You can get current connection with connected_node property

Returns nothing

del_job(**job_ids*)
Completely delete a job from a node.

Note that this is similar to FASTACK, but limited to a single node since no DELJOB cluster bus message is sent to other nodes.

Parameters **job_ids** –

dequeue(**job_ids*)
Remove the job from the queue.

Parameters **job_ids** – list of job_ids

enqueue(**job_ids*)
Queue jobs if not already queued.

Parameters **job_ids** –

execute_command(**args*, ***kwargs*)
Execute a command on the connected server.

fast_ack(**job_ids*)
Perform a best effort cluster wide deletion of the specified job IDs.

FASTACK jobid1 jobid2 ... jobidN

Parameters **job_ids** –

get_connection()
Return current connected_nodes connection.

Return type redis.Redis

get_job(*queues*, *timeout=None*, *count=None*, *nohang=False*, *withcounters=False*)
Return some number of jobs from specified queues.

GETJOB [NOHANG] [TIMEOUT <ms-timeout>] [COUNT <count>] [WITHCOUNTERS] FROM queue1 queue2 ... queueN

Parameters **queues** – name of queues

Returns list of tuple(job_id, queue_name, job), tuple(job_id, queue_name, job, nacks, additional_deliveries) or empty list

Return type list

hello()

Returns hello format version, this node ID, all the nodes IDs, IP addresses, ports, and priority (lower is better, means node more available). Clients should use this as an handshake command when connecting with a Disque node.

HELLO :returns: [<hello format version>, <this node ID>, [<all the nodes IDs, IP addresses, ports, and priority>, ...]

info()

Return server information.

INFO

Returns server info

jscan(*cursor=0*, *count=None*, *busyloop=None*, *queue=None*, *state=None*, *reply=None*)

Iterate all the existing jobs in the local node.

Parameters

- **count** – An hint about how much work to do per iteration.
- **busyloop** – Block and return all the elements in a busy loop.
- **queue** – Return only jobs in the specified queue.
- **state** – Must be a list - Return jobs in the specified state. Can be used multiple times for a logic OR.
- **reply** – None or string {“all”, “id”} - Job reply type. Type can be all or id. Default is to report just the job ID. If all is specified the full job state is returned like for the SHOW command.

nack_job(**job_ids*)

Acknowledge the failure of one or more jobs via job IDs.

NACK jobid1 jobid2 ... jobidN

Parameters **job_ids** – list of job_ids

pause(*queue_name*, *kw_in=None*, *kw_out=None*, *kw_all=None*, *kw_none=None*, *kw_state=None*, *kw_bcast=None*)

Pause a queue.

Unfortunately, the PAUSE keywords are mostly reserved words in Python, so I’ve been a little creative in the function variable names. Open to suggestions to change it (canardleteer)

Parameters

- **queue_name** – The job queue we are modifying.
- **kw_in** – pause the queue in input.
- **kw_out** – pause the queue in output.
- **kw_all** – pause the queue in input and output (same as specifying both the in and out options).
- **kw_none** – clear the paused state in input and output.

- **kw_state** – just report the current queue state.
- **kw_bcast** – send a PAUSE command to all the reachable nodes of the cluster to set the same queue in the other nodes to the same state.

qlen (*queue_name*)
Return the length of the named queue.

QLEN <qname>

Parameters **queue_name** – name of the queue

Returns length of the queue

qpeek (*queue_name*, *count*)
Return, without consuming from queue, count jobs.

If count is positive the specified number of jobs are returned from the oldest to the newest (in the same best-effort FIFO order as GETJOB). If count is negative the commands changes behavior and shows the count newest jobs, from the newest from the oldest.

QPEEK <qname> <count>

Parameters

- **queue_name** – name of the queue
- **count** –

qscan (*cursor=0*, *count=None*, *busyloop=None*, *minlen=None*, *maxlen=None*, *importrate=None*)
Iterate all the existing queues in the local node.

Parameters

- **count** – An hint about how much work to do per iteration.
- **busyloop** – Block and return all the elements in a busy loop.
- **minlen** – Don't return elements with less than count jobs queued.
- **maxlen** – Don't return elements with more than count jobs queued.
- **importrate** – Only return elements with an job import rate (from other nodes) >= rate.

qstat (*queue_name*, *return_dict=False*)
Return the status of the queue (currently unimplemented).

Future support / testing of QSTAT support in Disque

QSTAT <qname>

Return produced ... consumed ... idle ... sources [...] ctime ...

show (*job_id*, *return_dict=False*)
Describe the job.

Parameters **job_id** –

working (*job_id*)
Signal Disque to postpone the next time it will deliver the job again.

WORKING <jobid>

Parameters **job_id** – name of the job still being worked on

Returns returns the number of seconds you (likely) postponed the message visiblity for other workers

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

Index